ЛАБОРАТОРНАЯ РАБОТА Система управления версиями Git

ЦЕЛЬ РАБОТЫ

Изучение и получение навыков работы с системой управления версиями Git.

Введение

Система управления версиями (от англ. *Version Control System*, VCS или *Revision Control System*) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Такие системы наиболее широко используются при разработке программного обеспечения для хранения исходных кодов разрабатываемой программы. Применение систем управления версиями особенно актуально при коллективной разработке, когда изменения в общий программный код вносятся одновременно несколькими разработчиками и необходимо обеспечить их согласованную работу. Изучение и получение навыков работы с системой управления версиями является важным этапов в профессиональной подготовке любого программиста.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Введение

Git — это набор консольных утилит, которые отслеживают и фиксируют изменения в файлах (чаще всего речь идет об исходном коде программ, но его можно использовать для любых файлов). С его помощью можно откатиться на более старую версию проекта, сравнивать, анализировать, сливать изменения и многое другое. Этот процесс и называется контролем версий.

Git является распределенным, то есть не зависит от одного центрального сервера, на котором хранятся файлы. Вместо этого он работает полностью локально, сохраняя данные в папках на жестком диске, которые называются репозиторием. Тем не менее, можно хранить копию репозитория онлайн, это сильно облегчает работу над одним проектом для нескольких людей. Для этого используются сайты вроде github и bitbucket.

Настройка

После установки Git нужно изменить некоторые настройки. Настроим самые важные: наше имя пользователя и адрес электронной почты. Откройте консоль и запустите команды (где вместо «My Name» и myEmail@example.com задайте свои):

```
git config --global user.name "My Name"
git config --global user.email myEmail@example.com
```

После этого каждое действие будет отмечено именем и почтой. Таким образом, другие пользователи всегда будут в курсе, кто отвечает за какие изменения — это вносит порядок.

Создание репозитория

Git хранит свои файлы и историю прямо в папке проекта. Чтобы создать новый репозиторий, нужно открыть консоль, зайти в папку нашего проекта и выполнить команду **init**. Это включит

приложение в этой конкретной папке и создаст скрытую папку .git, где будет храниться история репозитория и настройки.

Создайте на рабочем столе папку под названием git_exercise. Для этого в окне консоли введите:

```
C:\Users\Роман>mkdir .\Desktop\git_exercise\

C:\Users\Роман>cd .\Desktop\git_exercise\
```

Замечание. Вместо «C:\Users\Роман» следует использовать путь текущего пользователя (далее эта подстрока будет скрыта).

Далее введите:

```
git init
```

Система выполнит команду и выведет результат:

```
Initialized empty Git repository in

C:/Users/Роман/Desktop/git_exercise/.git/
```

Это значит, что репозиторий был успешно создан и он пока пустой. Теперь создайте текстовый файл под названием hello.txt и сохраните его в директории git_exercise.

Определение состояния

status — это команда, которая показывает информацию о текущем состоянии репозитория. Запуск **git status** на нашем репозитории должен выдать:

```
git status

On branch master

No commits yet

Untracked files:

 (use "git add <file>..." to include in what will be committed)
hello.txt

nothing added to commit but untracked files present (use "git add"
to track)
```

Сообщение говорит о том, что файл hello.txt неотслеживаемый. Это значит, что файл новый и система еще не знает, нужно ли следить за изменениями в файле или его можно просто игнорировать. Для того, чтобы начать отслеживать новый файл, нужно его специальным образом объявить.

Подготовка файлов

В Git используется концепция области подготовленных файлов. Только подготовленные файлы могут отслеживаться на предмет изменений. Для того, чтобы файл стал отслеживаемым его нужно добавить в репозиторий. Делается это с помощью команды **add**:

```
git add hello.txt
```

Если нам нужно добавить все, что находится в директории, мы можем использовать команду **add -A**.

Повторна проверка статуса дает такой ответ:

```
git status
On branch master
No commits yet
Changes to be committed:
 (use "git rm --cached <file>..." to unstage)
 new file: hello.txt
```

Файл готов к коммиту. Сообщение о состоянии также говорит нам о том, какие изменения относительно файла были проведены в области подготовки — в данном случае это новый файл, но файлы могут быть модифицированы или удалены.

Коммит (фиксация изменений)

Коммит представляет собой состояние репозитория в определенный момент времени. Это похоже на снапшот (снимок), к которому можно вернуться и увидеть состояние объектов на определенный момент времени. Чтобы зафиксировать изменения, нужно хотя бы одно изменение в области подготовки, после которого можно коммитить:

```
git commit -m "Initial commit"
[master (root-commit) 9758809] Initial commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hello.txt
```

Команда **commit** создаст новый коммит со всеми изменениями из области подготовки (добавление файла hello.txt). Ключ **-m** и сообщение «Initial commit.» — это созданное пользователем описание всех изменений, включенных в коммит. *Замечание*. Считается хорошей практикой делать коммиты часто и всегда писать содержательные комментарии.

Удаленные репозитории

Выполненный коммит является локальным — существует только в скрытой папке .git на нашей файловой системе. Несмотря на то, что сам по себе локальный репозиторий полезен, в большинстве случаев мы хотим поделиться нашей работой или доставить код на сервер, где он будет выполняться.

1. Подключение к удаленному репозиторию

Чтобы загрузить что-нибудь в удаленный репозиторий, сначала нужно к нему подключиться. В этом руководстве будет использован адрес

https://github.com/RomanKovin/Some-Project.git, но вам необходимо создать **свой репозиторий** в GitHub, BitBucket или любом другом сервисе.

Чтобы связать локальный репозиторий с репозиторием на GitHub, необходимо выполнить команду **remote add origin** в консоли. Обратите внимание, что нужно обязательно **изменить** URI репозитория на свой!

```
git remote add origin https://github.com/RomanKovin/Some-
Project.git
```

2. Отправка изменений на сервер

Чтобы переслать локальный коммит на сервер используется команда **push**. Она принимает два параметра: имя удаленного репозитория (здесь это origin) и ветку, в которую необходимо внести изменения (master — это ветка по умолчанию для всех репозиториев).

```
git push origin master

Enumerating objects: 3, done.

Counting objects: 100% (3/3), done.

Writing objects: 100% (3/3), 215 bytes | 107.00 KiB/s, done. Total 3
(delta 0), reused 0 (delta 0), pack-reused 0

To https://github.com/RomanKovin/Some-Project.git

* [new branch] master -> master
```

Замечание. После ввода команды будет запрошен логин и пароль к сервису GitHub. Используйте свой собственный логин и пароль, который задавали при регистрации.

После успешной отправки нужно зайти через браузер на удаленный репозиторий https://github.com/RomanKovin/Some-Project и убедиться, что в него добавлен файл hello.txt

3. Клонирование репозитория

Другие пользователи GitHub могут просматривать ваш репозиторий. Они могут скачать из него данные и получить полностью работоспособную копию вашего проекта при помощи команды **clone**:

```
git clone https://github.com/RomanKovin/Some-Project.git
```

Новый локальный репозиторий создается автоматически с GitHub в качестве удаленного репозитория.

4. Запрос изменений с сервера

Если вы сделали изменения в вашем репозитории, другие пользователи могут скачать изменения при помощи команды **pull**:

```
git pull origin master

From https://github.com/RomanKovin/Some-Project

* branch master -> FETCH_HEAD

Already up to date.
```

Так как новых коммитов с тех пор, как был склонирован проект, не было, нет никаких изменений доступных для скачивания.

Ветвление

Во время разработки новой функциональности считается хорошей практикой работать с копией оригинального проекта, которую называют веткой. Ветви имеют свою собственную историю и

изолированные друг от друга изменения до тех пор, пока вы не решаете слить изменения вместе. Это может происходить по следующим причинам:

• уже рабочая, стабильная версия кода сохраняется;

• различные новые функции могут разрабатываться параллельно разными программистами;

• разработчики могут работать с собственными ветками без риска, что кодовая база поменяется из-за чужих изменений;

• в случае сомнений различные реализации одной и той же идеи могут быть разработаны в разных ветках и затем сравниваться.

1. Создание новой ветки

Основная ветка в каждом репозитории называется master. Чтобы создать еще одну ветку, используется команду **branch <name>**:

```
git branch some_feature
```

Это создаст новую ветку, точную копию ветки master. Повторный запуск команды покажет существующие ветки:

```
git branch
* master
 some_feature
```

Звездочкой показывается активная ветка.

2. Переключение между ветками

Для переключения на другую ветку используется команда **checkout**. Она принимает один параметр — имя ветки, на которую необходимо переключиться:

```
git checkout some_feature
Switched to branch 'some_feature'
```

3. Слияние веток

Наша “новая фича” будет еще одним текстовым файлом под названием feature.txt. Создадим его, добавим и коммитим:

```
git add feature.txt
git commit -m "New feature complete”
[some_feature 82cdab5] New feature complete
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 feature.txt
```

Изменения завершены, теперь можем переключиться обратно на ветку master.

```
git checkout master
Switched to branch 'master'
```

Если мы откроем проект в файловом менеджере, мы не увидим файла feature.txt, потому что мы переключились обратно на ветку master, в которой такого файла не существует. Чтобы он появился, нужно воспользоваться командой **merge** для объединения веток (применения изменений из ветки some_feature к основной версии проекта):

```
git merge some_feature
Updating 9758809..82cdab5
Fast-forward
feature.txt | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 feature.txt
```

Теперь ветка master актуальна. Если ветка some_feature больше не нужна, то её можно удалить:

```
git branch -d some_feature
Deleted branch some_feature (was 82cdab5).
```

Официальная документация по Git доступна по ссылке https://git-scm.com/doc.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Практическое задание

Для выполнения практической части работы используйте выданный преподавателем проект.

Необходимо решить следующие задачи:

1. Создать локальный репозиторий для проекта.

2. Добавить файлы с исходными кодами проекта в локальный репозиторий (master).

3. Создать удаленный репозиторий для проекта.

4. Добавить файлы с исходными кодами проекта в удаленный репозиторий.

5. Внести изменения в исходный код проекта (добавив любую другую функциональность).

6. Сохранить изменения в локальном репозитории в виде отдельной ветки «version 2».

7. Сохранить изменения в ветке «version 2» в удаленном репозитории.

8. Внести изменения ветки «version 2» в основную «master».

9. Проверить результат, пересобрав проект.

Список контрольных вопросов для самопроверки

1. В чем преимущества распределенных систем управления версиями по сравнению с централизованными?

2. Где хранятся версии?

3. Что такое репозиторий?

4. Что такое удаленный репозиторий?

ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать следующие разделы:

1. Титульный лист, оформленный согласно утвержденному образцу.

2. Цели и задачи выполняемой лабораторной работы.

3. Пошаговое описание выполняемых заданий лабораторной работы.

4. Ответы на контрольные вопросы.

5. Заключение.